CS 3410 Lab 5

Fall 2025

Agenda

1 Circuit Waveform Practice

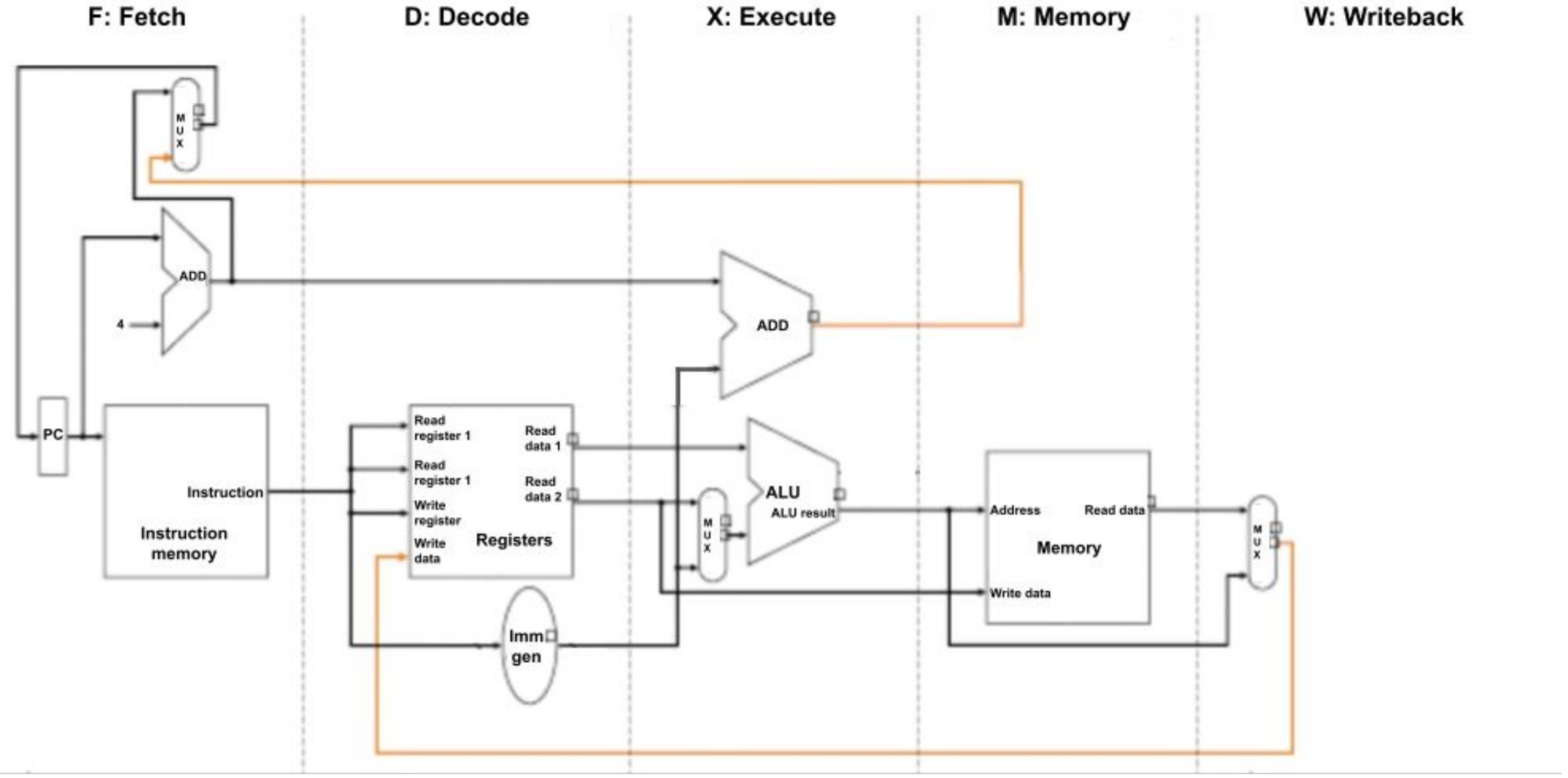
2 CPU Overview

3 Decoding and Encoding

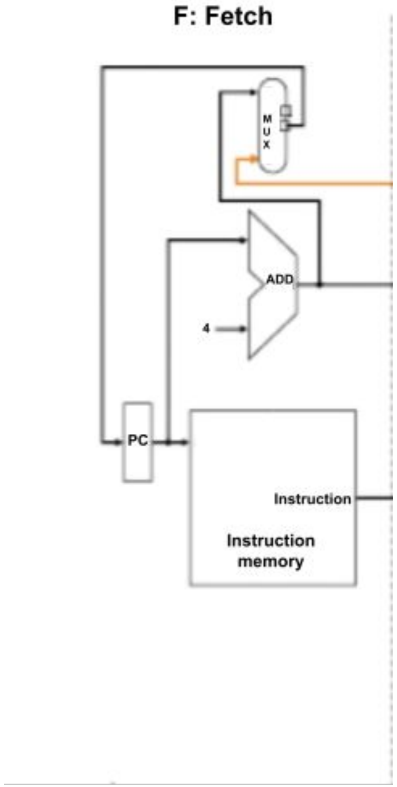
Circuit Waveform Practice

CPU Overview

A 5-Stage RISC Processor



Stage 1: Fetch



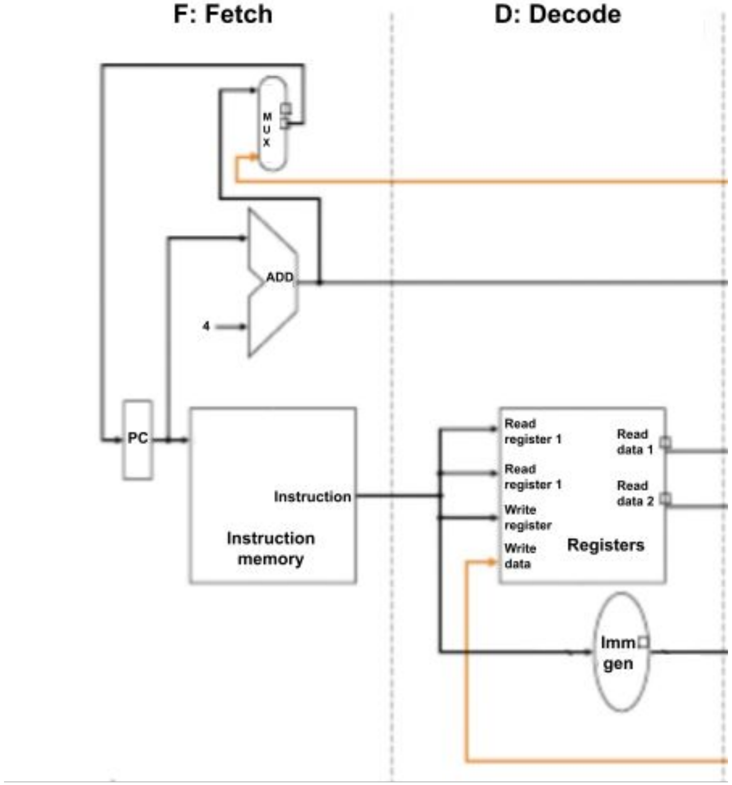
Instruction Memory:

- Stores instructions to execute on the CPU

Program Counter (PC):

- Points to the next instruction to execute
- Increments by 4 (1 word) to get next instruction
- Or by an immediate when we need to jump or branch

Stage 2: Decode



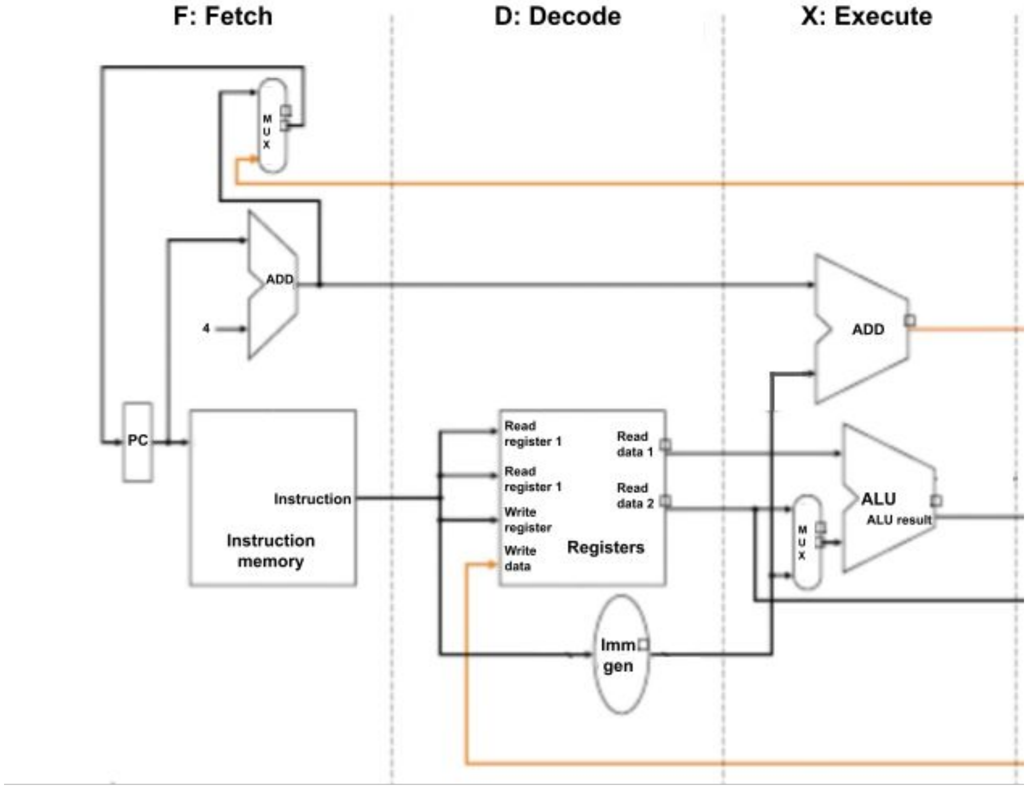
Decode parts of the instruction to decide how to execute it

Register File:

- Extract values for `rs1`, `rs2` and `rd` registers
- Extract immediate, sign-extend to convert to 64-bit number

Note: Not all instructions will use `rs1`, `rs2` or `rd` registers

Stage 3: Exec



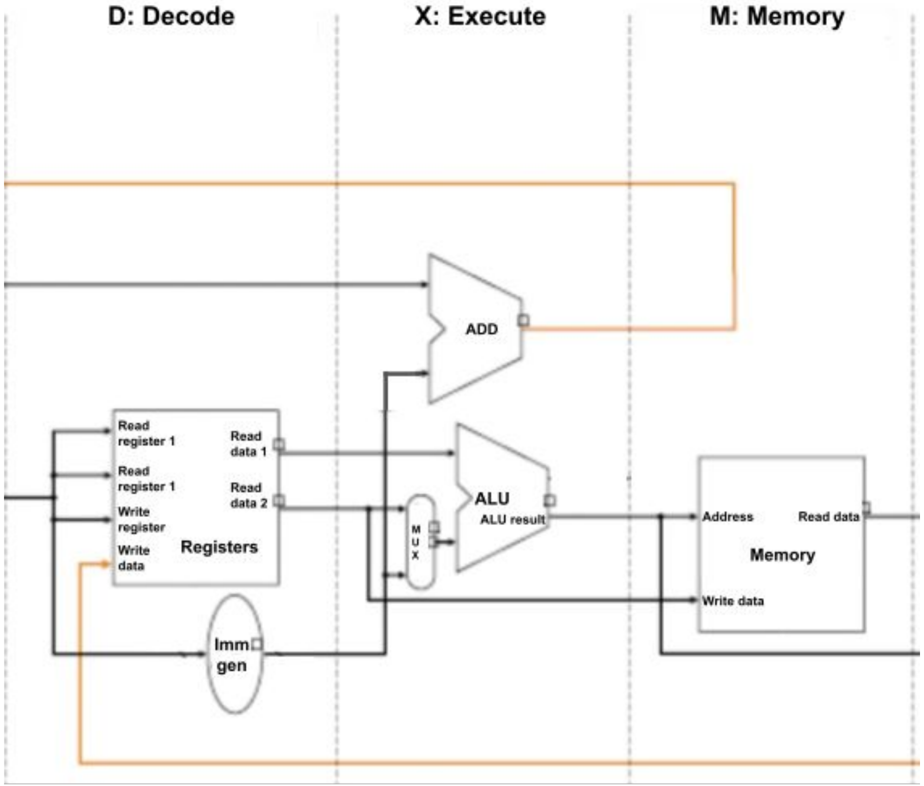
Adder (top):

- Calculate PC + offset. This may or may not be used depending on the instruction

ALU:

- Performs an operation using `rs1` and either `rs2` or `immediate`.
- Type of op also depends on instruction

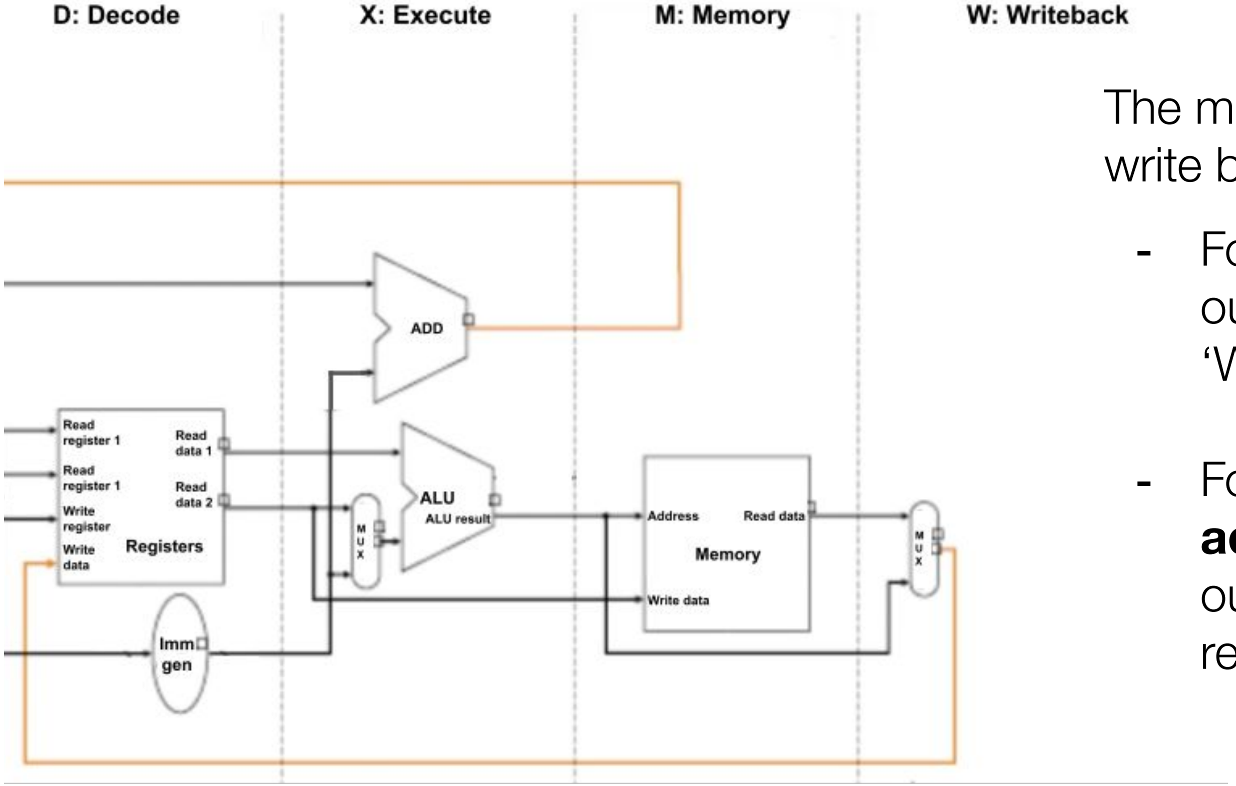
Stage 4: Memory



Data Memory:

- May carry out either a **read** or **write** operation, depending on instruction
- If **write**, this stage writes data from a register to a particular address
- If **read**, it reads from a particular location and forwards it to the next stage

Stage 5: Writeback



The mux decides what value to write back to the register file.

- For **load** instructions, write the output from memory to the ‘Write data’ register
- For arithmetic instructions (like **add**), write back the ALU output to the ‘Write data’ register

Note: Not all instructions will use this stage (think store instructions)

Decoding and Encoding

I-type Instructions

31 – 20	19 – 15	14 – 12	11 – 7	6 – 0
imm[11:0]	rs1	funct3	rd	opcode

- `addi` and `andi` are examples of I-type instructions

I-type Instructions

31 – 20	19 – 15	14 – 12	11 – 7	6 – 0
imm[11:0]	rs1	**funct3**	rd	**opcode**

- `addi` and `andi` are examples of I-type instructions

- We use the **opcode** and **funct3** (and sometimes funct7) bits of the instruction to decide how the instruction will execute

I-type Instructions

31 – 20	**19 – 15**	**14 – 12**	**11 – 7**	**6 – 0**
imm[11:0]	rs1	**funct3**	rd	**opcode**

- \`addi\` and \`andi\` are examples of I-type instructions

- We use the **opcode** and **funct3** (and sometimes funct7) bits of the instruction to decide how the instruction will execute

Instruction	**opcode**	**funct3**
`addi`	`0010011`	`000`
`andi`	`0010011`	`111`

Relevant opcode and funct3 bits for `addi` and `andi`

Some useful tips for the assignment

- Use the **`info` struct** to store metadata about your instruction, including what type it is. We provide a mapping from instructions to integers via the #define macros in sol.h.
- Think how the **`memory` stage** will be used in the lab **(hint: I-type instructions won't need read or write!)**. We should propagate some information to the Writeback stage to use the output from the ALU and not the Memory.
- After using the memory stage to send the information from execute to writeback, consider how your writeback stage should update the state of the program(the PC counter) to prepare it for the next instruction. (maybe **PC + 4**?)

How to test your implementation

- Make sure you clone the assignment repo and your terminal is under that folder
- To test your implementation for the assignment, follow the following steps:
  - Compile your implementation with “*rv make*”.
  - Write a test script in RISC-V assembly. We have an example test in the release file called check.s
  - Compile the test script using “*rv asbin check.s*”
  - Run the implementation using “*rv qemu runner < check.s*”
  - You can also initialize the registers, such as “*rv qemu runner 1@0x1 < check.s*”
- All these information can be found in the assignment instruction.